ADAPTACION DE PROCODI PARA COMPUTACION PARALELA

Universidad Nacional de La Matanza, Buenos Aires, Argentina
Florencio Varela 1903 - Teléfono: 4480-8900 Interno 8630

Mg. Daniel A. Giulianelli
dgiulian@unlam.edu.ar

Ing. Rocío A. Rodríguez
rrodri@unlam.edu.ar

Ing. Pablo M. Vera
pablovera@unlam.edu.ar

RESUMEN

UML (Lenguaje de Modelado Unificado) es un lenguaje amplio de propósito general, el cual ha sido originado como un lenguaje gráfico por excelencia. Sin embargo para modelar ciertas aplicaciones con características particulares el vocabulario gráfico de UML resulta ser muy reducido, requiriendo hacer uso de mecanismos de extensibilidad tal como estereotipos los cuales no tienen una construcción gráfica y por ende le restan claridad. Es por ello que se requiere de otros lenguajes como SYS ML (aprobado por el OMG Object Management Group) el cual cuenta con un vocabulario propio que extiende a UML para atender aquellas características no nomencladas por el mismo.

En éste paper se hará énfasis en la necesidad de ampliar el lenguaje de UML a fin de poder modelar aquellas características particulares de las aplicaciones con procesos concurrentes y distribuidos no nomencladas, a través de construcciones gráficas que conformen un vocabulario que permita extender a UML. Este conjunto de construcciones gráficas definen la especificación de un lenguaje que hemos denominado PROCODI (Lenguaje de extensibilidad para PROcesos COncurrentes y DIstribuidos), el que contempla todas las características de las aplicaciones con procesos concurrentes y distribuidos. Se presenta además la forma en que PROCODI logra modelar aplicaciones de computación paralela por medio de su adaptabilidad y vocabulario gráfico.

1. ANTECEDENTES

Con la necesidad de modelar aplicaciones con procesos concurrentes y distribuidos utilizando UML, se crea en el 2005 este equipo de investigación. A fin de difundir los avances de la presente tarea, se presentaron y expusieron papers en relevantes congresos nacionales e internacionales.

2. RECURSOS DE EXTENSIBILIDAD

UML es un lenguaje de propósito general el que no cuenta con la nomenclatura necesaria para todo tipo de modelado. En algunos casos particulares se hace necesario extender el vocabulario de UML para poder modelar características que no se encuentran nomencladas en el mismo, ésta es la razón por la cual UML propone mecanismos de extensibilidad los cuales permiten ampliar su vocabulario:

- Estereotipos: Permiten la creación de nuevos tipos de bloques de construcción que derivan de otros existentes pero no son específicos a un problema particular.
- Valores Etiquetados: Propiedades nuevas para elementos existentes.
- Restricciones: Forma de imponer reglas (de consistencia o de negocio) sobre elementos y sus propiedades.

Dichos mecanismos implican generalmente el agregado de frases, por ejemplo: propiedades, restricciones, etc. lo que se contrapone a las características gráficas del lenguaje. Es por ello que resulta conveniente crear construcciones gráficas para representar los estereotipos. Esta práctica es soportada por UML. Los autores de UML (Booch, Rumbaugh y Jacobson), para modelar el proceso unificado

utilizan todo tipo de mecanismos de extensibilidad incluso crean construcciones gráficas para algunas características no nomencladas [1]. Así mismo cabe destacar otras iniciativas como la de "SYS ML" que es un lenguaje de extensibilidad aprobado por la OMG, creado en el 2003 (para más información ver [6]). Al modelar aplicaciones con procesos concurrentes y distribuidos es necesario definir nomenclatura gráfica para ciertas características propias de este tipo de aplicaciones. Es por ello que se crea un lenguaje de extensibilidad al que se ha denominado PROCODI (PROcesos COncurrentes y DIstribuidos), el que tiene todo el vocabulario gráfico necesario y además una especificación de la forma en que es conveniente aplicarse. A lo largo de este paper se presentan las características no contempladas en UML de las aplicaciones con procesos concurrentes y distribuidos, asignándole a cada una de ellas una construcción gráfica a fin de presentar el vocabulario simbólico de PROCODI.

3. COMO SE GENERA PROCODI

Se basa en UML e incorpora nomenclatura para:

- Acceso a los recursos Compartidos (Semáforos, Monitores).
- Diferenciación entre mensajes y RPC (Remote Procedure Call – Llamadas a procedimientos remotos).
- Identificación de área de memoria compartida.
- Diferenciación entre RPC Sincrónicos y Asincrónicos.
- Cardinalidad para tareas que se harán en forma idéntica en distintos threads.
- Timer para aquellos casos en que las acciones repetitivas tienen un tiempo establecido de repetición.
- Manejo y control de excepciones
- Identificar procesos que requieren independencia en la ejecución paralela de manera de resaltar claramente aquellos procesos cuya ejecución no puede ser segmentada en distintos procesadores

Tomando los elementos propios del diagrama de actividades es posible hacer una adaptación, la cual permitirá:

- Diferenciar Nodos mediante las Calles del Diagrama de Actividades.
- Indicar Subcalles para distinguir threads.
- Establecer tipo de conexión entre los nodos.

4. EL VOCABULARIO SIMBOLICO DE PROCODI

A continuación se presenta el vocabulario gráfico, propio de PROCODI, que permite modelar aquellas características propias de los procesos concurrentes y distribuidos, no nomencladas en forma gráfica por UML.

- **Semáforos y monitores:** Cuando se utilizan hilos es muy posible que existan recursos que se deben compartir y por lo tanto es necesario administrar su acceso, ya que solamente un thread puede utilizarlo en un momento dado. Dos métodos comunes para la administración de recursos en un ambiente concurrente son los semáforos y los monitores [6].
  Proponemos construcciones gráficas para indicar recursos compartidos y el método de acceso a los mismos. Dentro de estas construcciones es posible especificar el tipo de recurso al que nos estamos refiriendo. Por otra parte también se propone diferenciar entre los semáforos binarios (aquellos que controlan el acceso a un sólo recurso) de los semáforos que controlan el acceso de n recursos. Pudiendo especificarse, en el caso de que sea necesario, la cantidad de recursos. La Figura 1, a la izquierda muestra un semáforo binario y a la derecha la representación gráfica para

un semáforo que maneja N recursos. La Figura 2 muestra la construcción gráfica adoptada para un monitor.

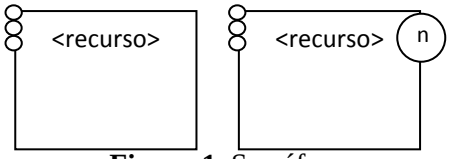

Figura 1. Semáforos

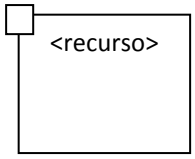

Figura 2. Monitor

- **Cardinalidad:** Cuando se cuenta con múltiples conexiones provenientes desde distintos nodos, generalmente existe un proceso principal que monitorea las conexiones y al llegar nuevas conexiones crea distintos hilos para cada una de ellas. Si las acciones que realizan cada uno de esos hilos son idénticas se propone englobarlas en la construcción gráfica mostrada en la Figura 3.

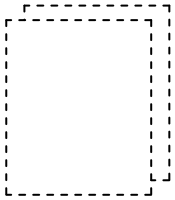

Figura 3. Cardinalidad

- **Timer:** Extendiendo la nomenclatura de cardinalidad es posible representar aquellas actividades que se realizan cada cierta cantidad de tiempo. Para esto se utiliza la construcción gráfica presentada en la Figura 4, siendo posible detallar el intervalo de tiempo en el cual se repetirá la rutina detallada dentro de esta construcción.
  Un ejemplo en donde puede observarse de forma rápida la necesidad de utilizar esta construcción es en el caso del chequeo de correo electrónico.

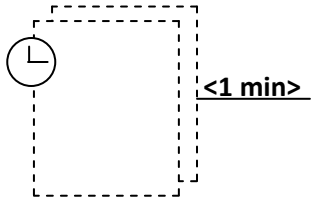

Figura 4. Timer

- **Mensajes y RPC:** El procesamiento distribuido requiere que los procesos alojados en distintos host, se comuniquen de alguna manera para poder intercambiar información. Existen dos formas distintas de realizar dicha comunicación: mediante el envío de mensajes y mediante la utilización de RPC. A su **vez** los mensajes pueden ser asincrónicos y sincrónicos. Un mensaje sincrónico obliga al emisor a esperar una respuesta antes de continuar con sus tareas, mientras que con el asincrónico se envía el mensaje y se continúa con el resto de las tareas. En cambio los RPC son en su mayoría llamadas sincrónicas ya que actúan como simples llamadas a procedimientos como si estuvieran en una misma computadora.
  Para modelar la comunicación entre los procesos se mantiene la nomenclatura habitual de UML donde un mensaje sincrónico se representa con una punta de flecha rellena y un mensaje asincrónico con una punta flecha abierta (es recomendable consultar el manual de especificación de UML 2.0 [4]). Respetando esta nomenclatura proponemos para mayor claridad diferenciar los RPC de los mensajes usando la nomenclatura mostrada en la Figura 5.

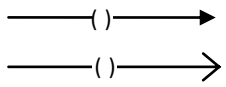

Figura 5. RPC Sincrónico y Asincrónico

- **Excepciones:** Para poder especificar las excepciones disparadas por los distintos procesos y como se manejan los mismos, se utiliza la propuesta de UML de modelado de excepciones en los diagramas de actividades (Figura 6). Para más información sobre las excepciones se recomienda ver [5]. Pero también es posible que el nodo que dispare la excepción y el nodo que contiene el manejador de la misma, no se conecten en forma directa, en ese caso proponemos especificar en los nodos intermedios que la recepcionan (sin atenderla) la derivación a otro nodo. Esto se indicará por medio de la construcción gráfica diseñada para utilizarse en los nodos intermedios (Figura 7).

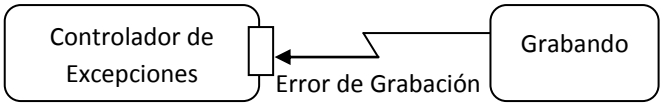

Figura 6. Controlador de excepciones

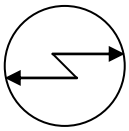

Figura 7. Reenvío de excepciones

5. ADAPTACIONES A DIAGRAMAS DE UML

A continuación se presentan recursos de modelado, que surgen de hacer adaptaciones a los elementos del diagrama de actividades. Este conjunto de recursos permitirán obtener un diagrama de actividades enriquecido con las características propias de las aplicaciones con procesos concurrentes y distribuidos.

- **Nodos:** Cada calle del Diagrama de Actividades será un nodo de la aplicación.
- **Threads:** El modelado del procesamiento concurrente puede requerir la diferenciación de los distintos hilos (threads) de ejecución del sistema. Se propone también para una notación más clara, cuando es necesario detallar varios estados dispares dentro de cada hilo, utilizar una notación de calles con líneas punteadas dentro de la calle principal del nodo, para indicar el procesamiento independiente de cada hilo (ver Figura 8).

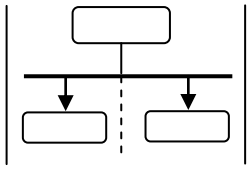

Figura 8. Subcalles para denotar hilos de Ejecución

- **Conexión entre nodos:** Los nodos que interactúan pueden tener diferentes tipos de conexiones físicas entre sí. Lo que **en** algunas ocasiones es importante destacar, ya que según sea el tipo de enlace habrá ciertas velocidades de transferencia que variarán. Con este fin se toma la notación del diagrama de despliegue y por medio de una línea que vincule los dos nodos en cuestión, se podrá aclarar el tipo de enlace (ver Figura 9).

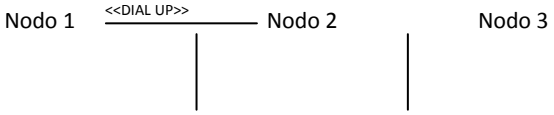

Figura 9. Conexión entre el nodo1 y el nodo 2

6. COMPUTACION PARALELA

Una computadora estándar ejecuta instrucciones de un programa en orden para producir un resultado. La computación paralela produce el mismo resultado utilizando múltiples procesadores a fin de obtener un menor tiempo de ejecución [2].

Cada una de las distintas computadoras, se modelan utilizando calles. Dentro de cada calle, a través de las subcalles, será posible modelar la distribución de porciones de código entre los procesadores de cada computadora.

PROCODI permite modelar las diferentes arquitecturas para memoria compartida, las cuales son requeridas en desarrollos planificados para computación paralela:

- Memoria compartida: una memoria única y global accesible desde todos los procesadores (ver figura 10).
- Memoria compartida – distribuida: la memoria está físicamente distribuida pero lógicamente compartida (ver figura 11)

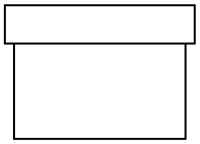

Figura 10. Memoria Compartida

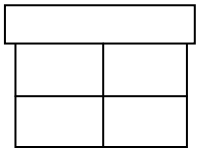

Figura 11. Memoria Compartida Distribuida

- Procesos independientes: Por otra parte existen procesos que deben resolverse en forma atómica, es decir no pueden ser divididos y ejecutados en partes distintas por diferentes procesadores. En estos casos la ejecución parcial de los mismos arrojaría un resultado distinto que su ejecución secuencial. Por ello, estos procesos deben ser manejados en forma secuencial en un procesador que trabaje independiente para los mismos. En la figura 12, se muestra un estado tradicional, el cual es posible procesarse en paralelo en (varios procesadores) y la construcción gráfica diseñada para un estado independiente.

Figura 12. A la derecha se muestra la gráfica para estados con posibilidad de procesarse en paralelo y a la izquierda el caso de estado independiente

7. CONCLUSIONES

Es importante notar que PROCODI se apoya en el lenguaje de UML extendiéndolo para aplicaciones con procesos concurrentes y distribuidos. Para agregar vocabulario faltante a UML, se toma como basamento la experiencia del equipo de trabajo, junto con la interacción entre pares concentrados en ésta disciplina y se modelan diversas aplicaciones con procesos concurrentes y distribuidos a fin de evaluar las limitaciones de UML.

Como PROCODI es un lenguaje basado en UML, no propone nuevos diagramas, sino que para se utilizan los elementos del diagrama de actividades de forma favorable para las aplicaciones con procesos concurrentes y distribuidos, agregándole al mismo todos los elementos del vocabulario de PROCODI a fin de modelar las características de éste tipo de aplicaciones.

Se trabaja actualmente en el modelado de diversas aplicaciones, con distintos grados de complejidad, desarrolladas para computación paralela, tomando en cuenta la posibilidad de mantener procesos secuenciales y ejecutar en forma paralela aquellos en los que no varié por ello su resultado. Es por ello que en este paper se presentan elementos de PROCODI vinculados con el diseño de aplicaciones para computación paralela, dejando en evidencia, a través de calles y subcalles que es posible modelar aplicaciones de este tipo.

8. REFERENCIAS

1) Booch G, Rumbaugh J y Jacobson I. *El proceso unificado de desarrollo de software*. Addison Wesley, 2001.
2) Grama Ananth, *Introduction to Parallel Computing*, Addison Wesley, 2003.
3) OMG, *SYS ML Versión 1.0*, Septiembre 2007. http://www.omg.org/docs/formal/07-09-01.pdf
4) OMG, UML, OMG Unified Modeling Language, Infrastructure, Versión 2.1.2. Noviembre 2007, Disponible en: http://www.omg.org/docs/formal/07-11-04.pdf
5) Randy Miller, *What's New in UML 2? Model Exceptions*, Junio, 2003. Disponible en: http://dn.codegear.com/article/30169
6) Stalling W. *Operating Systems Internals and Design Principles*. Third Edition, Prentice Hall, 1998.